Ubiquitous Computing

Computer technology continuously advances and it enters people's lives more and more as they get better and we can perform certain tasks faster with them, or can perform certain tasks automatically. The existence of computers is due to people because people created them and people are using them. Computer technology and computers probably wouldn't have a meaning in whatever they do without people's needs for them either directly or indirectly. We may benefit from computers directly when their benefits are directly due to them such as using a word processor to accomplish our tasks (e.g. writing a report or paper), while we can benefit indirectly when we *eventually* get benefits due to having computers processing tasks for us but the connection is not that obvious. For instance, some products IKEA sells are cheaper due to computers because with the help of computers certain tasks in production require less effort.

Humans and computers together could be viewed as a *system*, where they affect one another. Ubiquitous computing[1] (ubicomp, for short, as Weiser uses it in [18]) is about computers everywhere surrounding humans, communicating with each other and interacting with people in certain ways. One of the main goals of ubiquitous computing is that it needs to be *distraction-free.* The research group for the Aura project [14] developed at Carnegie-Mellon University (CMU) claims the importance of this as follows "The most precious resource in a computer system is no longer its processor, memory, disk or network. Rather, it is a resource not subject to Moore's law: *User Attention.*" in [15]. Russell and Weiser also agree that user attention is the scarcest resource [11].

Computing is a technology and according to Mark Weiser, who is known as the father of ubiquitous computing due to initiating research work at Xerox Palo Alto Research Center (PARC) in 1988 [18], the most profound technologies are those that disappear [21] as he states in his seminal paper. According to him, "writing" is already a technology that disappears. We no longer think about the symbols, or concentrate on the shapes of letters. It is natural for us to write as we think. The details of writing, or the overhead, which are the selection of symbols, their shape, position all are irrelevant to the task, namely to the thoughts that we are going to express through writing. Writing is so pervasive; it is everywhere, shop names, on the books, on some sticky note that is attached to a wall, on papers, books, and magazines, even in small things such as candy wrappers. Its prevalence is due to its profoundness, it offers so much than it requires and this is largely because the irrelevant details to the task "disappear" (writing can probably be stated to be an *automatic process* [2] that consumes no attentional resources of a person after enough practice).

Mark Weiser contrasts the technology of writing with that of computers at then (1991) and states that computers are in a world on their own and makes an analogy to the era when writing was a bulky process as in order to write, there were scribes that would make ink and bake clay. Weiser notes that the "disappearance" concept also exists in other famous people's minds in different terms: nobelist Herbert A. Simon calls it "compiling"; philosopher Michael Polanyi calls it the "tacit dimension"; psychologist J.J.Gibson calls it "visual invariants"; and philosophers Gadamer and Heidegger call it the "horizon" and "the ready to hand" [21]. Weiser believes that computers should also vanish in the background

[1] also known as Pervasive Computing

allowing people to concentrate on their tasks and sets it as the goal of the ubiquitous computing, the third wave of the computing revolution [19]. The first and second waves, or *trends* [20] are mainframe computing, and personal computing. The mainframe era has many situations where there is a single computer and many people using it, while in the personal computing era, the current era, every person has one or two computers and in the era of ubiquitous computing, there is one person benefiting from many computers. Weiser predicts that the use of personal computers will increase before decreasing, while we enter the era of ubicomp, short for ubiquitous computing. He claims that computers are already around us, although small or specialized, in watches, toasters, ovens, cars and wallets. The difference will be that they will be communicating with each other for the benefit of people, and don't need to be more powerful in order to do many tasks at once.

Making an analogy with kitchen utensils, a general computer that does almost every task for us is similar to having one apparatus in the kitchen that can cook, boil, chop, toast, and wash, which would probably be complicated, heavy, and hard to use [19]. With this analogy, Weiser supports his idea of "less being an improvement" as the computing devices will be small and do only a few functions but very well. Jain and Wullert et al. [5] also point to the importance of using less (material and energy), and discuss about the life span of the devices, using them longer and how to construct devices (such as labeling components with RF tags to record their identities) for smart disposal.

The power of ubiquitous computing doesn't come from the mere collection of the many individual computing devices but arises from the interconnection of them. They will be able to send information to each other and use local information to customize according to people's current needs.

Early Prototypes

One of the early research efforts for ubiquitous computing came from the Olivetti Research group (ORL) in England and Xerox PARC, namely to build the active badge location system [17]. The active badge is a small device that can be worn by personnel in a building. Every active badge has a unique identity information on it and it transmits this identity information via InfraRed (IR) every 10 seconds. The active badge can be used indoors only because the IR is to be received by the sensors which are located in a building equipped with them. Some of the early scenarios that were thought for the uses of active badge location systems are that doors could open on the right badge wearer, rooms could greet people by their names, phone calls can be automatically forwarded to the room that the person called is in (by looking up its id in the system and locating the person), and computer terminals could retrieve user preferences automatically [21].

The Xerox PARC research group believed that two issues were important: location and scale. Active badges were an example to address the location issue. The scale is about the size of the computing devices. The research group in Xerox PARC developed 3 types of prototypes of varying scale, namely tabs, pads and boards. These are general names. The research group in Xerox PARC produced a prototype for each called ParcTab, Mpad, and Liveboard, respectively [18]. Tabs are inch-scale, pads are foot-scale, while boards are yard-scale. It was envisioned that a room would contain approximately 100 tabs, 10-20 pads and 1 or 2 boards.

Tabs are small enough to be attached to things, they can be in various forms, such as an active post-it note. pads behave like a sheet of paper, they are not a laptop or notebook computer though.

Pads are designed more to be "scrap" computers. They don't have a static identity and accommodate various tasks. For instance, one could use pads to organize tasks on a desk, as a reminder for each task (or part of task) that needs to be done. Unlike laptops, they are not intended for carrying. One may leave a pad in a room, and pick up another one in another room, and not carry pads from room to room. Laptops have a too small visual area, a desk of 9"x11" usually is not sufficient, one needs more space, which can be achieved by placing many pads next to each other.

Boards are usually intended to cover large areas such as a wall. Interactive operations such as a wireless electronic chalk, using which one can write without touching on the board [21]. Some possible uses of boards are to electronically mediate meetings, even share meetings across the Atlantic (PARC (in US) and EuroPARC (in Europe) had such experiences). A board where public information is displayed on may adapt the information presented according to its readers. This can be achieved by having users wear active badges on and put sensors on or near the board. This is an example for the emerging power of the ubiquitous computing devices due to their *interaction*.

Early thoughts for the design of ubiquitous computing

The conventional operating systems usually assume that the hardware and software configurations won't change. However, this is not true for ubicomp devices which change locations and situations, and need to adapt to new circumstances where the configurations need to be altered. To support ubicomp devices, the future operating systems had better shrink and grow to fit needs, dynamically. Rashid (CMU) and Tanenbaum (VU) designed a micro-kernel operating system that uses scaffolding, where modules can be added and removed (for this reason the kernel is minimal, and is called *micro*) [21].

From the hardware point of view, small devices need to consume little power to avoid frequent replacement of batteries which will mean interruption for users. Since power is proportional to the square of voltage, it is effective to reduce power by reducing voltage. While much effort has been spent to reduce the size of chips, for low power devices, by doubling the size of the chip, the clock speed can be reduced in half, which helps decrease the power requirements. Since circuits in chips designed for high speed generally fail to work at low voltages, it is not possible to simply design the fastest possible chip and then run it at a reduced clock speed and voltage [18].

The research group in Xerox PARC, when they designed their ParcTab system, tried to have the batteries last longer by having the system enter power saving mode after a brief period of inactivity [13]. The active badge system used a light-dependent component, and the badge would turn off when it gets dark to conserve battery life [17]. Both systems used IR because infrared transducers are small, low cost, and most importantly low power.

According to Weiser, three types of network connections would become prevalent: tiny-range connections, long-range wireless connections, and very high speed wireless connections [21].

A potential problem would be privacy [19], because so much personal information would be around, in the computing devices and in the networks that connect them. EuroPARC applied the idea that personal information should be only stored in a computer

which belongs to the person as opposed to storing at a central repository, which was the earlier approach used [18]. Other systems that need personal information would get it from that computer. This way the information would be secured in a place that is personal and would be protected by whatever precautionary or protective software has been installed on that particular computer.

New Views in Ubiquitous Computing

The interactivity of ubiquitous computing devices to adapt to various situations necessitates different kind of principles when building applications [4]. Banavar, from IBM T.J. Watson Center, provides us with different views for three ubicomp elements. (1) A *device* is not a repository of custom software, but a portal into an application or data space. (2) An *application* is not software to exploit device's capabilities but is a means for a user to perform a task. (3) The *computing environment* is not a virtual space that exists to store and run software, rather it is a user's information enhanced surroundings [3]. Tasks need to be dynamic in ubicomp so that they can easily be transferred from device to device [10] as in the scenario with Jane in [4], where she continuous to monitor the meeting from her PDA and when she leaves, from her cell phone, and then from the limousine's back seat's screen while going to the airport. If the device is viewed as a repository of custom software, then all problems with porting the software to another device arise. A solution to this is to view the devices as a portal to an application, so it communicates with the application and retrieves whatever necessary and possible to a currently used device. In the same scenario above, while the limousine drives, different network connections are formed (seamlessly) and the quality of those may be different. The resolution of the video streaming adapts to the quality of the network connections available at that time. This is an excellent example to the view that the application should be viewed as a means to perform a task as the important thing for Jane is to continue getting information about the meeting. If the application was written to exploit the device's capabilities only, then maybe it would use the highest resolution being insensitive to the quality of network connections available, and Jane wouldn't be able to perform her task satisfactorily.

According to the new application model Banavar talks about, the lifecycle of pervasive computing can be divided into three: design-time, load-time and run-time [3].

Design Time

This is the time when the developer creates, maintains and enhances the application. When the application is designed, no specific assumptions should be made about the user interface. Even the assumption that there is a screen, will be problematic when the application should run on a voice synthesizer, or a phone. The user interaction should not be hard coded, but dynamically created according to the structure of the task. Tasks could be defined in terms of subtasks but not in terms of static interaction elements such as a button or a spoken command. When it comes to take an input, either a button or a spoken command should be able to be given as the situation permits. In general, abstract interaction elements should be identified that captures user intent.

The services that the application uses need to be specified in an abstract way and not named explicitly. The design should be general enough so that even services not known at design time could be used by the application later at run-time as appropriate. A location service is a case in point. Specifically, a location service will be used according to

availability such as GPS, a web based location service, or another mechanism. An abstract service description language may be needed to achieve this.

Load Time

During load time, the system composes, adapts and loads the application components into an application instance on particular hardware devices. Since devices should be viewed as portals to applications, they need to discover what applications are available and the system needs to adapt the application to the resources available in the device. The application should be specified in terms of requirements, and the resources in terms of capabilities such that a mediating algorithm between the two can negotiate an appropriate match.

The system must be dynamic at load time because the tasks my depend on the surroundings. It should discover and compose the services that are available as software components that *live* in the surrounding environment. In contrast to today, software needs to manually be installed from a CD to device, for instance, and therefore is not dynamic. One effort considered detaching context handling from service matters with the use of context-aware packets (CAPs) in an architecture called CAPEUS (Context-Aware Packets Enabling Ubiquitous Services) [12].

There may not be enough resources available. In that case, only hostable applications and services need to be displayed when the user needs to make a choice. At times, due to limited resources or low performance of the device at hand, it is desirable to split the execution burden between the device and available servers. This operation is called *appartiotioning* [3].

There are several challenges related to the negotiation, which is necessary to match available services with available resources. One challenge is modeling device characteristics and application requirements. A metric needs to be developed and the requirements and resources should be specified using the same terms. Another challenge is to find fast and efficient appartitioning algorithms. One reason for this is that loading may not be a one-time operation but may be performed one after another as some resources become unavailable and some others available, and reappointments are necessary.

Run Time

During run-time, the user invokes the application and uses its functionality. At run-time, resources need to be monitored so that the application can adapt according to the resources available. Hand-off of task context from one environment to another, such as from office to car, should be supported. The user's access to the task should not be interrupted. This could be achieved by maintaining the state as continuous, or if that is not possible, provide off-line functionalities that the user can continue working on the task.

Handling of unexpected failures such as exhausting batteries or service crashes is highly desirable as these have great impact on the user at least in terms of system availability.

At run-time, reappointments are necessary when disconnection occurs. The reappointments need to be non-obtrusive to the user. Reappointments had better be done when the changes are relatively permanent and not transient. Ideally, the system should detect the transient changes and not make unnecessary reappointments. Reappointments could be user-initiated, too, and will be performed when the user expects a disconnection to

occur, for example. When there are multiple choices, it may also be desirable in certain cases to make the choices available to the user to let the user have control over the choice.

According to the connectivity, the appointer may migrate the execution burden, such as migrating to client if there are sporadic network disconnection. A major challenge is to automate disconnections and reconnections as much as possible. Mechanisms such as queuing network requests need to be designed and provided as a framework to be used when disconnections occur. Disconnections are expected phenomena, and should not be confused with failures, but often it is challenging to distinguish between them for the system. It is necessary for the system to detect it correctly because depending on whether it is a disconnection or failure the system may need to behave differently. Checkpointing strategies may be adapted as a way of handling failures. Replication is a commonly used method in traditional systems for fault tolerance but it is not sufficient by itself for ubicomp applications [10].

Raatikainen, Christensen and Nakajima discuss application requirements for middleware, which is a set of generic services above the operating system [10]. Typical middleware services include location transparency and failure transparency. Common Object Request Broker Architecture (CORBA), Java 2™ Enterprise, Micro Editions (J2EE & J2ME), Distributed Common Object Model (DCOM), and Wireless Application Environment (WAE) are examples of middleware [10]. In order to provide applications that are context-aware, personalized and adaptive, various software architectures are proposed (e.g. Sahara and MITA [10]). These architectures define an *execution support layer* that encapsulates the functions of middleware. An execution support layer should provide fast service development and deployment. It should be easy to divide the application logic, distribute these components and configure them as well as redistribute and reconfigure them. It should provide a consistent, efficiently accessible, reliable and highly available information base, probably a distributed, and replicated worldwide "file system" that also supports intelligent synchronizations of data after disconnections (except being worldwide, Coda [16] has most of these features). Several research issues are addressed in [10]. One of the research challenges is the partitioning of applications and placement of different co-operating parts. Moreover, when the execution environment changes significantly, the co-operating parts may need to be redistributed.

Ubiquitous Computing Projects & Applications

There are several ubiquitous computing projects developed by the systems group of Mahadev Satyanarayanan at CMU such as Coda [16], Aura [14], and Odyssey. Coda is a distributed file system that provides disconnected operation for mobile computing. It is high performance through client side caching. It has good scalability, a well defined semantics of sharing, even in the presence of network failures. It continues to operate during partial network failures in a server network and it supports server replication. In addition, it has a security model for authentication, encryption and access control. Coda works very well when there are few conflicts and its strategy to resolve conflicts is pretty effective for objects with simple semantics such as file directories, but is ineffective when the objects are complex and many conflicts (30%) may remain unsolved [9]. The DAgora

system proposes a more effective solution based on a caching mechanism in clients and a weakly consistent server replication strategy [9].

The goal of the Aura project [15] is to design the system in such a way that it doesn't distract the user. It tries to provide *an invisible halo of computing and information services* that persists regardless of location [4].

Some other projects address the following challenges: Cooltown of HP, service construction and composition, as well as user experience validation; Jini of Sun, context-based adaptation; Oxygen at MIT and Aware Home at Georgia Tech, user experience enhancement and validation; PIMA at IBM, application development and deployment as well as context based adaptation; Portolane at University of Washington, software infrastructure, service construction and composition, as well as user experience enhancement and validation; and Semantic Web addresses semantic modeling. [4]

Semantic modeling tries to solve the interoperability problem that arises due to the very large range of devices. The current interoperability standards won't scale to satisfy ubicomp interoperability issues due to dramatic increase of connectivity requirements. Semantic modeling is also motivated from the critical observation that the source of most serious challenges in pervasive computing are not technological but structural [7]. Four categories of semantic conflicts are reported in [7]: naming conflicts (different names used to represent the same concepts), domain conflicts (different reference systems, such as different currency), structural conflicts (different data organizations that represent the same structure), and metadata conflicts (the same concept is represented with one data type within the modeling type of one system and with a different type within another system). In the past, these conflicts are solved by designing the system appropriately, which turns out to be inappropriate for systems that need to interchange in dynamic environments. Ontologies are provided as a solution by introducing a level where systems use common ontologies to decide about semantics. With the use of ontologies, semantic interoperability can be achieved by runtime comparison of and inference about ontological information [7]. Several XML based representation languages are emerging that are based on W3C's Resource Description Framework (RDF). Using semantic mark up languages and ontologies, knowledge about various pervasive computing parties can be encoded at creation time and reasoned upon later. Since it is very likely that there will be no common agreement on one representation language, each ubicomp element must encode knowledge about itself using explicit reference to an appropriate ontology.

Researchers at Siemens work to build a situated computing framework [8]. Small screen devices may not have all the resources for some applications such as multimedia. They propose outsourcing and redirection as possible solutions. For example, consider that there is a speakerless monitor and a telephone in a room. A person's PDA may outsource the video part of AV to the monitor and the audio part to the telephone. At times, conversion may be another solution when there is no resource for a certain data. For example, when there is text, but only audio resources are available, the text may be converted to audio form and presented as such. They categorize the interactivity into three as abdicative, cooperative and exclusive. In abdicative case, the PDA hands over the control to the output device, while in exclusive case the only input device is PDA, which is used when the output device has no input facilities available, such as a TV. In cooperative case, PDA and input capabilities of the output device can jointly be used to control the application. A special mobile user interface on the PDA is needed for the cooperative and

exclusive modes. One possible application scenario is controlling the mouse cursor wirelessly via the PDA. In such a scenario, Myers et al. proposes to enter all mouse commands on the PDA using stylus [6]. This is a possible solution provided that there is wireless communication ability [8].

Pervasive computing was needed for the International Monitoring System of the CTBTO PrepCom[2], an organization based on the Comprehensive Nuclear Test Ban Treaty (CTBT) for knowledge management. There are many IMS stations that are interconnected, and there are CTBTO manuals that provide procedures for various issues such as installation, data quality, data transmission, up-time, and security. However, a lot of tacit knowledge and information needs to be used which is discovered and doesn't exist in the manuals. The station operators need such information, especially for some special purpose equipment for an IMS station because it takes a lot of effort to learn it by trial and error. Pervasive computing could capture and disseminate such local knowledge in order to assist station operators. [1] discusses all the problems and how a ubiquitous framework helps to solve those.

Conclusion

Ubiquitous computing, if finally realizable, seems to have so much to offer to our lives as envisioned by researchers. However, it also seems to be a large, complicated and multilevel problem with many different aspects that it is even hard to define what ubicomp is in terms of an architecture. There have been such efforts but those architectures seem not to be general enough to cover all aspects. It is such an open ended problem and it is likely that some definitions include several high expectations from ubicomp that some of them may not be achievable even in principle. There has been much research done for ubiquitous computing and it seems there is some accumulated experience today about it as compared to when it first started but there is still a long way to go to attain real ubiquitous computing. It is also not clear whether a unified ubiquitous computing will be achieved in the future or at least different parts that can communicate with each other. And maybe we will have many different ubicomp frameworks that are not compatible with each other and maybe we will call them something else but not ubiquitous computing. Nevertheless, the motivation is clear to ubicomp researchers, and it is expected that ubicomp efforts will continue at least during the near future.

References

1. P. Amann and G. Quirchmayr. Foundation of a framework to support knowledge management in the field of context-aware and pervasive computing. In *Proceedings of the Australasian information security workshop conference on ACSW frontiers 2003*, pages 119-131. Australian Computer Society, Inc., 2003
2. Anderson, J. R. *Cognitive Psychology and Its Implications.* Freeman, 4th Edition, page 92, 1995. ISBN: 0-7167-2385-9.
3. G. Banavar, J. Beck, E.Gluzberg, J. Munson, J. Sussman, and D. Zukowski. Challenges: an application model for pervasive computing. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 266-274, ACM Press, 2000.

[2]http://www.ctbto.org

4. G. Banavar and A. Bernstein. Software infrastructure and design challenges for ubiquitous computing applications. *Commun. ACM*, 45(12):92-96, 2002.
5. R. Jain and J. Wullert, II. Challenges:: environmental design for pervasive computing systems. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 263-270. ACM Press, 2002.
6. B.A.Myers, H. Stiel, and R. Gargiulo. Collaboration using multiple PDAs connected to a PC. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 285-294. ACM Press, 1998.
7. D. O'Sullivan and D. Lewis. Semantically driven service interoperability for pervasive computing. In *Proceedings of the 3rd ACM international workshop on Data engineering for wireless and mobile access*, pages 17-24. ACM Press, 2003.
8. T.-L. Pham, G. Schneider, and S. Goose. A situated computing framework for mobile and ubiquitous multimedia access using small screen and composite devices. In *Proceedings of the eighth ACM international conference on Multimedia*, pages 323-331. ACM Press, 2000.
9. N. Preguiça, J.L. Martins, H.J. Domingos, and J. Simao. Flexible data storage for mobile computing. In *Proceedings of the 1999 ACM symposium on Applied computing*, pages 405-407. ACM Press, 1999.
10. K. Raatikainen, H. B. Christensen, and T. Nakajima. Application requirements for middleware for mobile and pervasive systems. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):16-24, 2002.
11. D.M. Russell and M. Weiser. The future of integrated design of ubiquitous computing in combined real & virtual worlds. In *CHI 98 conference summary on Human factors in computing systems*, pages 275-276. ACM Press, 1998.
12. M. Samulowitz, F. Michahelles, and C. Linhoff-Popien. Adaptive Interaction for Enabling Pervasive Services. In *Proceedings of the 2nd ACM international workshop on Data engineering for wireless and mobile access*, pages 20-26. ACM Press, 2001.
13. B. N. Schilit, N. Adams, R. Gold, M. M. Tso, and R. Want. The PARCTAB mobile computing system. In *Workshop on Workstation Operating Systems*, pages 34-39, 1993.
14. S. M. Thayer and P. Steenkiste. An architecture for the integration of physical and informational spaces. *Personal Ubiquitous Comput.*, 7(2):82-90, 2003.
15. The Aura Project, Carnegie-Mellon University. http://www.cs.cmu.edu/~aura/
16. P. J. Braam, The Coda Distributed File System. *Linux Journal*, 46-51, June 1998
17. R. Want, A. Hopper, V. Falcao, and J. Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91-102, 1992.
18. M. Weiser. Some computer science issues in ubiquitous computing. *Commun. ACM*, 36(7):75-84, 1993.
19. M. Weiser. The future of ubiquitous computing on campus. *Commun. ACM*, 41(1) 41-42, 1998.
20. M. Weiser. http://www.ubiq.com/hypertext/weiser/UbiHome.html, *Ubiquitous Computing*
21. M. Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3(3):3-11, 1999.